REMEMBERING WHAT TO ACT ON: OBJECT-CENTRIC EXECUTION STATE FOR MOBILE GUI AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-horizon mobile GUI tasks require agents to process multiple data objects across pages and applications. For reliable execution, agents must retain the required values while tracking how task data are organized, what has been done to each object, and which operations the task constraints allow. In this paper, we introduce **DOMES**, a method for **D**ynamic **O**bject-centric **M**aintenance of **E**xecution **S**tate, which unifies data retention and progress tracking by binding each object's structured data to its evolving processing status and task constraints. We train our agent with an online trajectory synthesis pipeline that combines template-based task expansion, automatic environment initialization, and programmatic completion verification. This enables automated collection of successful trajectories that jointly supervise state updates and GUI actions. Considering that state maintenance does not benefit every task, we introduce **STR-GRPO**, which contrasts rollouts with and without state maintenance on the same task while penalizing non-empty updates. This encourages the agent to maintain state only when it contributes to task success, while improving generalization to unseen tasks. To evaluate agents' ability to complete required data operations and respect task scope, we introduce **DataScope**, a scalable online mobile benchmark. Its parameterized templates vary the number of target objects and out-of-scope distractors while preserving workflow logic and providing executable environment initialization and final-state verification. Since binary success does not capture partial completion or scope violations, we propose app-level and data-level metrics to measure workflow progress and the completeness and precision of data operations, respectively. Experiments show that our 8B agent achieves a 76.7% success rate on AndroidWorld, outperforming UI-TARS-2-230B by 3.4 percentage points. On DataScope, it consistently outperforms the strongest same-sized baseline across app-level, data-level, and binary task-success metrics. Compared with standard GRPO, STR-GRPO substantially reduces state-update rates while maintaining comparable observed task success. We will release our agent and curated benchmark.

1 INTRODUCTION

Long-horizon mobile GUI tasks require agents to process multiple data objects across pages and applications (Kong et al., 2025; Rawles et al., 2024). Subsequent operations often depend on information encountered many steps earlier and no longer visible on the current screen (Chai et al., 2025; Chen et al., 2024; Yang et al., 2025; Yan et al., 2025). Agents must therefore retain the relevant data while tracking which operations have been completed and which remain necessary. Losing required data can prevent subsequent operations, while unclear processing states can lead to repeated work, omissions, or premature termination.

Recent methods support information retention and progress tracking through memory modules, progress summaries, and explicit state mechanisms (Xiao et al., 2026; Zhang et al., 2025; Wang et al., 2025b; Ye et al., 2025; Zheng et al., 2026; Shi et al., 2026a; Tian et al., 2026). However, retained values alone do not indicate when or how they should be used, while progress descriptions that are not linked to individual objects require agents to reconstruct these associations from memory or interaction history. Reconstructing these associations from long interaction histories is difficult, leaving agents uncertain about which objects still require action. We observe that task data, represented through their structure, values, and processing states, provide a natural basis for integrating information retention and progress tracking. By binding each object's values and relationships to its evolving processing

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

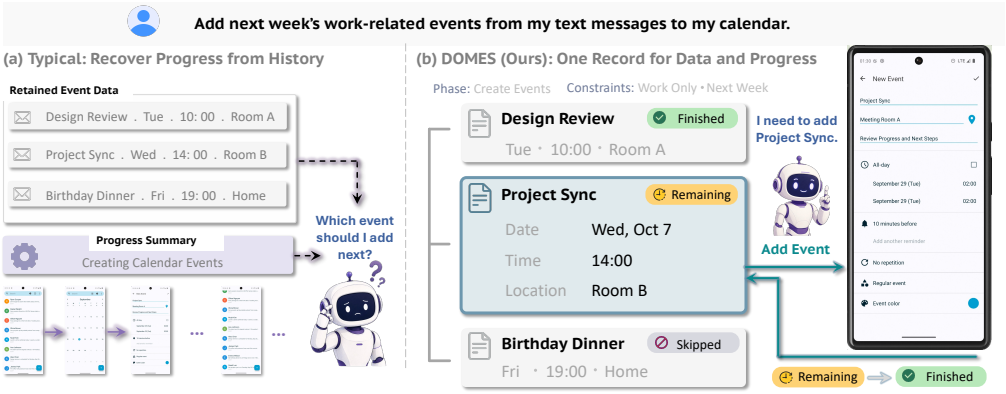

Figure 1: **DOMES unifies task data and execution progress through object-centric state. (a)** Without explicit links between retained data and progress summaries, agents must reconstruct object-level progress from interaction history, risking missed or repeated operations. **(b)** DOME tracks the workflow phase and constraints while binding each event's values to its processing state: finished, remaining, or skipped. Each record stores data, and its state updates upon confirmed completion.

state, the same representation preserves the information needed for execution and identifies the remaining work under task constraints, as illustrated in Figure 1 (a–b).

Building on this observation, we introduce **DOMES**, an object-centric execution-state representation that uses the objects being processed to organize both retained information and task progress. DOMES combines a common structure with a task-specific schema defining the objects and fields needed for execution. At the workflow level, it records the current phase, task constraints, and progress across collections of objects. At the object level, it binds each item's values to its processing state and preserves its relationship to the corresponding collection. This organization connects workflow progress with the status of individual objects, allowing the agent to identify remaining work using the retained data and task constraints. The agent uses DOMES alongside the current GUI observation and interaction history to select actions and update its execution state.

To teach our agent **DOMES-UI** how to construct and update DOMES during execution, we collect training trajectories that pair GUI actions with execution-state updates. Existing collection methods rely on labor-intensive human annotation (Li et al., 2024a) or automated exploration (Sun et al., 2025; Ramrakhya et al., 2026; Cheng et al., 2026), requiring further annotation or exploration to broaden task coverage. To generate additional task instances with verifiable completion conditions, we develop an online synthesis pipeline using parameterized templates. Each template jointly specifies the task goal, environment initialization, and programmatic completion checks, allowing us to vary task data while keeping instructions, initial states, and expected results consistent. A teacher agent executes the generated tasks, producing DOMES updates alongside GUI actions. We retain only trajectories whose final environment states pass verification and use them for supervised fine-tuning.

Although SFT teaches DOMES-UI how to maintain execution state, it does not explicitly optimize when the benefits justify the maintenance cost. We therefore introduce **STR-GRPO**, an online reinforcement learning method that jointly optimizes task completion and selective state maintenance. STR-GRPO contrasts DOMES-enabled and -disabled rollouts from the same task initialization and combines task-completion rewards with a penalty on nonempty updates. This objective encourages the agent to retain useful execution information while reducing unnecessary state maintenance.

To evaluate whether agents complete required data operations and respect task scope, we introduce **DataScope**, a scalable online mobile benchmark with 128 task instances. Its parameterized templates define interdependent operations and object relationships, with executable environment initialization and final-state verification. Within each template, we vary the number of target objects, associated records, and out-of-scope distractors while preserving workflow logic, enabling controlled evaluation as data-processing and filtering demands increase. Since binary task success alone does not capture partial completion or scope violations, we introduce two complementary metrics. **App-Level Progress** measures completion of required application subgoals, while **Scope-Aware F1** measures the completeness and precision of data operations.

Experiments demonstrate DOMES-UI's effectiveness on standard mobile tasks in AndroidWorld, its generalization to unseen environments in MobileWorld, and its ability to execute long-horizon data-processing workflows in DataScope. DOMES-UI-8B achieves a 76.7% success rate on AndroidWorld,

exceeding UI-TARS-2-230B by 3.4 percentage points. Among the evaluated agents trained without exposure to MobileWorld environments, DOMES-UI-8B achieves the highest success rate at 23.3%. On DataScope, it outperforms all evaluated end-to-end baselines across difficulty levels on SR, App-Level Progress, and Scope-Aware F1. Compared with standard GRPO, STR-GRPO reduces nonempty update rates by 13.8 and 14.6 points on AndroidWorld and MobileWorld, respectively, while maintaining comparable task success. Our contributions are summarized as follows:

- We propose **DOMES**, an object-centric execution-state representation that binds structured task data to processing states and constraints to guide mobile GUI execution.
- We develop a trajectory synthesis pipeline with task-completion verification for joint state and action supervision, and introduce **STR-GRPO** to learn selective state maintenance through rollout comparisons and maintenance penalties.
- We introduce **DataScope**, a scalable online mobile benchmark with configurable data, programmatic verification, and metrics for workflow progress and operation scope.

2 Related Work

Mobile GUI Agents and State Maintenance. Mobile agents increasingly support multi-step tasks across applications (Zhou et al., 2025; Xu et al., 2026; Qin et al., 2025; Ye et al., 2025). Memory supports execution by retaining the data values needed for later operations and a reusable interaction experience (Wen et al., 2024; Li et al., 2024b; Lee et al., 2024; Wang et al., 2025b). Recent methods also explicitly maintain task state. TSR tracks requirements, subgoal progress, and action feedback (Zheng et al., 2026), AgentProg organizes context through program variables and belief states (Tian et al., 2025), and ASM maintains causally linked intermediate-state anchors (Shi et al., 2026b). However, retaining a value does not establish whether it still requires processing, while progress summaries may leave individual objects' completion states unclear. DOMES brings data retention and progress tracking together by associating each actionable data unit's values and structure with its processing state under task constraints. We train agents to update this representation, preserving the information needed for subsequent operations while making the remaining work explicit.

Mobile Agent Benchmarks. Mobile GUI benchmarks can be broadly divided into offline datasets and online interactive environments. Offline benchmarks measure action prediction on recorded trajectories (Rawles et al., 2023), but cannot assess how errors affect subsequent execution. Online benchmarks support interactive execution and environment-based verification (Rawles et al., 2024; Xu et al., 2025; Kong et al., 2025), with further work examining intermediate progress and memory dependencies (Chai et al., 2025; Shi et al., 2026b). However, aggregate results do not isolate how data workloads and distractors affect execution completeness. DataScope supports scalable, data-centric evaluation through extensible templates that generate cross-application tasks on demand. By varying target objects, associated records, and distractors while preserving workflow logic, it enables controlled evaluation as processing demands increase. Its app-level and data-level metrics complement binary success by measuring partial progress, missed targets, and out-of-scope modifications.

3 Methods

As shown in Figure 2, we first introduce DOMES for jointly maintaining task data and execution progress (Sec. 3.1). We then describe verified trajectory synthesis for learning state updates and GUI actions (Sec. 3.2), followed by STR-GRPO for optimizing task completion and state-maintenance cost (Sec. 3.3). Finally, we present DataScope and its app-level and data-level metrics for evaluating data-centric workflows (Sec. 3.4).

3.1 DOMES: Object-Centric Execution State

DOMES jointly maintains task data and execution progress through a shared base structure with task-specific fields. As illustrated in Figure 2 (a), it combines workflow-level progress and constraints with the contents and processing states of actionable data units. An actionable data unit is the smallest semantically complete item treated as an operation target in the task, such as an expense record containing `expense`, `amount`, and `category`. Formally, the execution state at step t is

$$m_t = (\phi_t, \Sigma_t, E_t), \tag{1}$$

where ϕ_t contains workflow progress and task constraints, Σ_t defines the task-specific schema, and E_t contains data units indexed by stable identifiers.

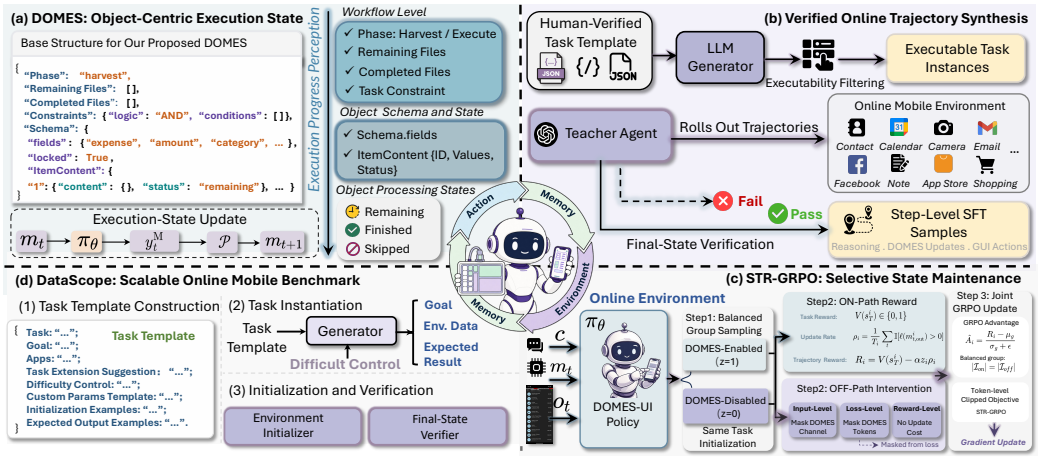

Figure 2: **Overview of Our Methodology.** **(a)** DOMES binds object values to processing states within a task-specific schema, alongside the workflow phase and task constraints. **(b)** Teacher rollouts on template-instantiated tasks are retained after final-state verification to supervise DOMES updates and GUI actions. **(c)** STR-GRPO contrasts balanced DOMES-enabled and DOMES-disabled rollouts from the same task initialization and penalizes nonempty updates to encourage selective state maintenance. **(d)** DataScope varies target and same-schema distractor counts and uses calibrated verifiers to measure task success, app-level progress, and data-level precision and completeness.

Workflow-Level State. The `Phase` field distinguishes `harvest`, during which the agent collects task-relevant data, from `execute`, during which it performs operations on the collected units. When source tracking is required, `RemainingFiles` and `CompletedFiles` record sources awaiting inspection and those already inspected. Otherwise, both remain empty. The agent transitions to `execute` after collecting sufficient information and inspecting all required sources. `Constraints` stores instruction-derived conditions, such as date ranges or category restrictions, that determine which units require processing.

Data-Unit Structure and State. The agent constructs `Schema.fields` to specify the attributes required for each actionable unit. It can add task-specific fields during construction, after which `Schema.locked` fixes the field set to prevent structural drift. Within this schema, `ItemContent` associates each unit's identifier with its structured values and processing status $s_i \in \{\texttt{remaining}, \texttt{finished}, \texttt{skipped}\}$. These statuses indicate whether the unit awaits processing, has completed its required operations, or should be excluded. Whereas `Constraints` records task-wide eligibility rules, `skipped` records the decision for a particular unit based on information acquired during execution. This organization preserves operation-relevant values together with the state needed to determine remaining work.

State-Conditioned Interaction. Before each model call, the current DOMES state m_t is inserted into the system prompt. Given task instruction c, current GUI observation o_t, interaction history $h_{<t}$, and m_t, the agent's response generation is formulated as:

$$y_t \sim \pi_\theta(\cdot \mid c, o_t, h_{<t}, m_t), \qquad (m_{t+1}, \rho_t, a_t) = \mathcal{P}(y_t), \tag{2}$$

where $\mathcal{P}$ extracts the updated state m_{t+1}, reasoning trace ρ_t, and executable action a_t. The updated state is carried into the next model call, allowing the agent to maintain task data and processing progress as interaction proceeds. The execution prompt is provided in Appendix E.1.

3.2 Verified Online Trajectory Synthesis

As shown in Figure 2 (b), we synthesize trajectories that jointly supervise DOMES updates and GUI actions through online interaction in a mobile environment with 20 applications.

Executable Task Generation. We manually design and verify 120 task templates, each with an environment initializer and a programmatic task-completion verifier. The initializer sets up task-relevant data and distractors in the required applications, while the verifier checks whether the final environment state satisfies the task goal. Each template also provides goal patterns, data-field specifications, examples, and guidelines for generating task variations. Guided by these templates, an LLM-assisted generator produces task instructions and matching initialization data. We generate

approximately 1.2K candidate tasks and retain 1.1K executable instances after filtering out malformed, invalid, or unreachable tasks.

Trajectory Collection and Supervision. For each task, a teacher framework combines an LLM planner (OpenAI, 2025) with UIIns (Chen et al., 2025) to generate grounded GUI actions. At each step, it produces a DOMES update, a reasoning trace, and an action, leaving DOMES empty when state maintenance is unnecessary. We retain trajectories that pass task-completion verification, yielding 21,713 step-level SFT samples. Each sample maps the context $(c, o_t, h_{<t}, m_t)$ to a teacher response containing (m_{t+1}, ρ_t, a_t), jointly supervising DOMES updates and action prediction.

3.3 STR-GRPO FOR LEARNING STATE MAINTENANCE

Since not every task requires state maintenance, we introduce **STR-GRPO** to learn when maintaining DOMES is worth its cost. It computes group-relative advantages over DOMES-enabled and DOMES-disabled rollouts of the same task, using task-completion rewards adjusted for maintenance cost.

State-Channel Intervention. For each task q, we sample a group of N_g rollouts from the rollout policy $\pi_{\theta_{\text{old}}}$, starting from the same initial environment state. Half enable DOMES and half disable it. Let $z_i \in \{0, 1\}$ indicate whether DOMES is enabled for rollout i. The policy input at step t is

$$x_t^i = (c, o_t^i, h_{<t}^i, \widetilde{m}_t^i), \qquad \widetilde{m}_t^i = \begin{cases} m_t^i, & z_i = 1, \\ \varnothing, & z_i = 0, \end{cases} \tag{3}$$

where c is the instruction for task q. In enabled rollouts, the current DOMES state is inserted into the system prompt and updated across steps. In disabled rollouts, the prompt instructs the agent not to use DOMES, and a rule-based check clears any DOMES block it nevertheless produces before the next step. Both conditions retain interaction history, including observations, actions, and reasoning.

Reward and Group-relative Advantage. For rollout i with T_i steps, the reward is formulated as:

$$R_i = V(s_{T_i}^i) - \alpha z_i u_i, \qquad u_i = \frac{1}{T_i} \sum_{t=1}^{T_i} \mathbb{I}[m_{t,\text{out}}^i \not\equiv \varnothing], \tag{4}$$

where $V(s_{T_i}^i) \in 0, 1$ verifies task completion from the final environment state. The generated DOMES block $m_{t,\text{out}}^i$ is considered empty when all its fields are empty, so u_i measures the fraction of steps with nonempty DOMES outputs. The coefficient $\alpha > 0$ controls the maintenance cost, applied only to enabled rollouts through z_i. We compute group-relative advantages jointly across both conditions through $\widehat{A}_i = \frac{R_i - \mu_g}{\sigma_g + \epsilon_A}$, where μ_g and σ_g are the group reward mean and standard deviation, and ϵ_A ensures numerical stability. Successful rollouts without DOMES receive higher rewards than those with nonempty outputs, encouraging task completion with less state maintenance.

Policy Optimization. The clipped optimization objective is formulated as:

$$\mathcal{L}_{\text{STR}}(\theta) = -\mathbb{E}\left[\sum_{(i,t,k)\in\mathcal{I}} \min\left(r_{i,t,k}\widehat{A}_i, , \text{clip}(r_{i,t,k}, 1-\varepsilon, 1+\varepsilon)\widehat{A}_i\right)\right], \tag{5}$$

where the expectation is over sampled rollout groups and ε is the clipping threshold. The token probability ratio is $r_{i,t,k} = \pi_\theta(y_{t,k}^i \mid x_t^i, y_{t,<k}^i)/\pi_{\theta_{\text{old}}}(y_{t,k}^i \mid x_t^i, y_{t,<k}^i)$, where $y_{t,k}^i$ denotes the k-th response token at step t of rollout i. The index set $\mathcal{I}$ includes DOMES, reasoning, and action tokens for enabled rollouts, but only reasoning and action tokens for disabled rollouts.

3.4 DATASCOPE ONLINE MOBILE BENCHMARK AND METRICS

Benchmark Overview. We introduce **DataScope**, a scalable online benchmark for data-centric mobile workflows. Each task requires agents to process all objects satisfying the task constraints while leaving out-of-scope distractors unchanged. These distractors share the targets' field schema or data type, requiring agents to distinguish objects by their values and task relevance. Parameterized templates vary target and distractor counts while preserving workflow structure and operation types, enabling controlled evaluation as data-processing and filtering demands increase.

For evaluation, we instantiate 32 manually designed templates at four difficulty levels, DC-V0 through DC-V3, yielding 128 task instances across 14 apps. Figure 3 (b) illustrates the increasing execution demands across these levels, with the distractor-to-target ratio rising from $1.03\times$ in DC-V0 to $2.47\times$ in DC-V3. The templates support generating additional task variants beyond this evaluation set.

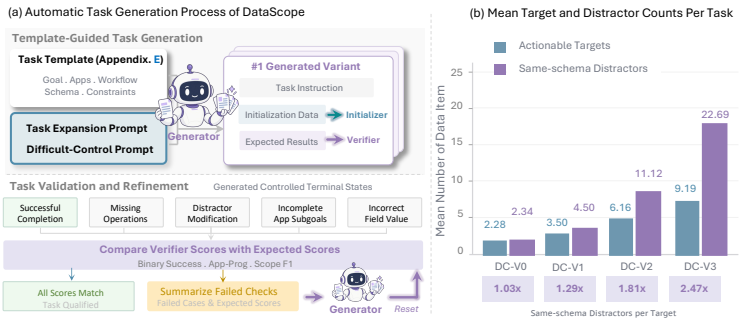

Figure 3: **Scalable task generation and controlled difficulty in DataScope. (a)** Template-guided generation produces executable task variants with controllable difficulty. Verifiers are tested on controlled terminal states and regenerated using feedback from failed checks. **(b)** The subsets reflect progressively increasing task-execution difficulty.

Task Construction and Verification. Figure 3 (a) summarizes the construction process. We manually design and verify templates specifying goals, apps, data schemas, object relationships, initialization examples, and expected outputs. A task expansion prompt guides the generation of new variants, while a difficulty control prompt specifies target and distractor counts. Using these templates and prompts, an LLM-assisted generator (OpenAI, 2025) produces consistent task instructions, initialization data, and expected outputs. The initializer writes the generated data into app storage, and the expected outputs specify the required changes and the data that must remain unchanged.

For each generated task, the generator constructs a programmatic verifier using the initializer, expected outputs, and calibration cases. Each case specifies a controlled terminal state and its expected task-success, App-Prog, and Scope-F1 scores. The cases cover completed tasks, missed operations, modified distractors, incomplete app subgoals, and incorrect field values. We load each state into the environment and compare the verifier's scores with the expected scores. Failed cases and score differences are summarized and fed back to the generator, which regenerates the verifier and reruns all checks. A task is retained only when its verifier returns the expected scores for every calibration case. Further details and examples appear in Appendices B, B.3, and B.4.

App- and Data-level Metrics. Binary task success indicates whether a task is completed, but does not reveal how much required work remains or whether the agent modifies out-of-scope data. We therefore introduce *App-Level Progress* (App-Prog) to measure the fraction of completed app-level subgoals and *Scope-Aware F1* (Scope-F1) to evaluate the coverage and precision of data operations.

For task q, let $\mathcal{A}_q$ denote the required apps and $c_{q,a} \in \{0, 1\}$ indicate whether all required operations in app a are complete. App-Prog is defined as:

$$S_{\text{prog}}(q) = \frac{1}{|\mathcal{A}_q|} \sum_{a \in \mathcal{A}_q} c_{q,a}. \tag{6}$$

Scope-F1 evaluates the completeness and precision of data operations using verification units, each identified by a data entry and, where applicable, a field and its value. For app a, let $\mathcal{D}^{\star}_{q,a}$ denote the required units and $\hat{\mathcal{D}}q, a$ the units realized by the agent, including incorrect and out-of-scope results, as determined from the final environment state. Their F1 score is $Fq, a = 2|\hat{\mathcal{D}}q, a \cap \mathcal{D}^{\star}q, a|/(|\hat{\mathcal{D}}q, a| + |\mathcal{D}^{\star}q, a| + \epsilon)$, where ϵ ensures numerical stability. Missing required results and unintended modifications both reduce this score. To account for modifications outside the required workflow, we average over the union of required apps $\mathcal{A}_q$ and apps modified by the agent $\mathcal{A}_q^{\text{act}}$:

$$S_{\text{scope}}(q) = \frac{1}{|\mathcal{A}_q \cup \mathcal{A}_q^{\text{act}}|} \sum_{a \in \mathcal{A}_q \cup \mathcal{A}_q^{\text{act}}} F_{q,a}. \tag{7}$$

4 Experiments

4.1 Experimental Setup

Models and Training. We initialize DOMES-UI from Qwen3-VL-4B and Qwen3-VL-8B (Bai et al., 2025) and train it through SFT followed by online RL. Both verified demonstration collection and online RL use an Android environment with 20 apps. Unless otherwise specified, ablations use the 8B backbone. For a controlled comparison, GRPO and STR-GRPO start from the same SFT checkpoint

Table 1: Performance on **AndroidWorld**. SR denotes success rate (%). Green/purple marks the best/second-best results among comparable models. Standard deviations are over three runs.

Method	Params	SR
Larger and proprietary models for reference		
UI-TARS (Qin et al., 2025)	72B	65.9
UI-Venus (Gu et al., 2025)	72B	65.9
Qwen3-VL-235B-A22B (Bai et al., 2025)	235B	63.7
UI-TARS-1.5 (Seed, 2025b)	–	64.2
Gemini-2.5-Pro (DeepMind, 2025)	–	69.7
Seed1.8 (Seed, 2025a)	–	70.7
UI-TARS-2 (Wang et al., 2025a)	230B	73.3
Recent 4B/8B models		
Step-GUI (Yan et al., 2025)	8B	67.7
MAI-UI (Zhou et al., 2025)	8B	70.7
GUI-Owl-1.5-Thinking (Xu et al., 2026)	8B	71.6
UI-Venus-1.5 (Team et al., 2026a)	8B	**73.7**
DOMES-UI (Ours)	4B	65.5 ± 0.9
DOMES-UI (Ours)	8B	76.7 ± 1.7

Table 2: Generalization on **MobileWorld**. We compare only agents trained without exposure to MobileWorld environments. Green/purple marks the best/second-best results. DOMES-UI-8B reports mean $\pm$ standard deviation over three runs.

Method	Params	SR
Larger models for reference		
UI-Venus-1.5 (Team et al., 2026b)	30B	17.1
UI-Venus (Gu et al., 2025)	72B	16.4
GUI-Owl (Ye et al., 2025)	72B	8.5
Recent 4B/7B/8B models		
GUI-Owl (Ye et al., 2025)	7B	7.7
UI-Venus (Gu et al., 2025)	7B	8.5
GELab-Zero (Team, 2025)	4B	16.1
DOMES-UI (Ours)	4B	20.5 ± 1.7
DOMES-UI (Ours)	8B	23.3 ± 0.9

Table 3: Evaluation of end-to-end mobile agents on **DataScope**. GPT-5.6 / GPT-5.2 + UIIns + DOMES serve as the framework-based references. SR denotes the binary success rate.

Method	Params	Data-Scope-V0			Data-Scope-V1			Data-Scope-V2			Data-Scope-V3		
		SR	S_{prog}	S_{scope}	SR	S_{prog}	S_{scope}	SR	S_{prog}	S_{scope}	SR	S_{prog}	S_{scope}
Agentic framework for reference													
GPT-5.6+ UIIns + DOMES	–	43.8	59.9	64.1	46.9	59.4	62.8	28.1	44.3	56.1	21.9	31.8	42.7
GPT-5.2+ UIIns + DOMES	–	21.9	27.1	28.3	12.5	25.0	20.3	6.2	16.1	21.3	3.1	9.3	13.2
End-to-end mobile agents													
GELab-Zero (Team, 2025)	4B	3.1	6.8	9.1	0.0	0.1	3.6	0.0	3.6	6.4	0.0	2.6	4.4
UI-TARS-1.5 (Seed, 2025b)	7B	0.0	1.6	2.2	0.0	1.6	2.0	0.0	1.6	1.8	0.0	1.6	1.5
GUI-Owl (Ye et al., 2025)	7B	3.1	6.2	7.4	**3.1**	4.7	7.2	0.0	1.6	1.1	0.0	1.6	1.5
UI-Venus (Gu et al., 2025)	7B	0.0	2.6	4.8	0.0	1.6	3.8	0.0	1.6	0.6	0.0	0.0	0.7
GUI-Owl-1.5 (Xu et al., 2026)	8B	**6.2**	**12.0**	**15.8**	**3.1**	**9.6**	**11.2**	0.0	2.6	**7.4**	0.0	2.6	**5.0**
MAI-UI (Zhou et al., 2025)	8B	3.1	8.9	10.1	0.0	3.1	4.5	**3.1**	**4.2**	3.9	0.0	**3.7**	4.2
GUI-Owl (Ye et al., 2025)	32B	3.1	4.7	7.4	0.0	4.2	4.9	0.0	1.6	1.1	0.0	0.0	0.5
UI-Venus-1.5-A3B (Team et al., 2026b)	30B	0.0	3.1	5.4	0.0	3.1	4.6	0.0	1.6	1.8	0.0	1.6	1.5
Our model													
DOMES-UI (Ours)	**8B**	**9.4**	**18.2**	**25.9**	**6.2**	**11.7**	**15.7**	**6.2**	**8.9**	**13.0**	**3.1**	**6.3**	**8.6**

and share RL tasks, rollout budgets, and optimization steps. STR-GRPO uses a maintenance-cost coefficient of $\alpha = 0.05$. We detail the infrastructure and hyperparameters in App. A.4 and A.3.

Evaluation Protocol. We evaluate on AndroidWorld (AW) (Rawles et al., 2024), the unseen MobileWorld (MW) environment (Kong et al., 2025), and DataScope. We report task success rate (SR), App-Prog, and Scope-F1 on DataScope. DOMES-UI uses greedy decoding with at most 2,048 generated tokens per interaction step. AndroidWorld follows its official task-specific step limits, and MobileWorld uses the official 50-step limit. On DataScope, all agents share task initializations, step budgets, termination conditions, and programmatic evaluation criteria, detailed in Appendix A.5.

Baselines. We evaluate both end-to-end mobile agents and execution-state methods implemented within agent frameworks. For end-to-end comparisons, we compare DOMES-UI with representative mobile agents, including MAI-UI (Zhou et al., 2025), GUI-Owl (Ye et al., 2025; Xu et al., 2026), UI-Venus (Gu et al., 2025; Team et al., 2026a), UI-TARS Seed (2025b), and GELab-Zero Team (2025). For framework-level comparisons, we reproduce the agent frameworks of ASM (Shi et al., 2026a) and AgentProg (Tian et al., 2026), retaining their execution workflows and state-management mechanisms while standardizing the planner and executor to GPT-5.2 and UIIns. All methods are evaluated under identical action spaces, decoding settings, step budgets, and evaluation criteria.

4.2 Main Results

We first evaluate DOMES-UI against end-to-end mobile agents on AndroidWorld, MobileWorld, and DataScope, then compare agent frameworks using the same planner and executor models.

Performance on Standard Benchmarks. Tables 1 and 2 report performance on AndroidWorld and MobileWorld. On AndroidWorld, DOMES-UI-8B achieves 76.7% SR, outperforming GUI-Owl-1.5-Thinking-8B by 5.1 percentage points and UI-TARS-2-230B by 3.4 points. DOMES-UI-4B reaches 65.5%, within 0.4 points of the 72B UI-TARS-SFT and UI-Venus models. To assess generalization to unseen environments, our MobileWorld comparison includes only agents trained without exposure to its environments. DOMES-UI-8B achieves the highest SR among these agents at 23.3%, exceeding

Table 4: Execution-state ablation on Qwen3-VL-8B-Instruct. Status-free retains task data and constraints but removes execution-status information. The length-matched control rewrites status-free reasoning and memory to match DOMES's supervised token count. All variants share trajectories and action supervision. Values are success rates (%).

Representation	SFT	+ STR-GRPO
AndroidWorld		
Status-free	**64.7**	**68.1**
Status-free (length-matched)	63.8	67.2
DOMES	**70.7**	**76.7**
MobileWorld		
Status-free	**14.7**	**18.1**
Status-free (length-matched)	13.8	17.2
DOMES	**17.2**	**23.3**

Table 5: Training ablation on Qwen3-VL-8B. Trained variants report mean ± standard deviation over three seeds. Update rate is the percentage of steps with nonempty DOMES outputs. Green/purple marks the best/second-best values according to the indicated directions.

Training	SR (%) ↑	Update (%) ↓
AndroidWorld		
SFT	70.7 ± 2.6	48.2 ± 1.5
SFT + GRPO	74.7 ± 1.3	45.7 ± 0.8
SFT + STR-GRPO	76.7 ± 1.7	31.9 ± 1.0
MobileWorld		
SFT	17.2 ± 2.6	49.6 ± 1.7
SFT + GRPO	20.7 ± 0.9	44.7 ± 0.9
SFT + STR-GRPO	23.3 ± 0.9	30.1 ± 1.0

UI-Venus-1.5-30B by 6.2 points. DOMES-UI-4B also surpasses all listed baselines at 20.5%. These results demonstrate strong execution performance and generalization to unseen environments.

Performance on DataScope. As shown in Table 3, DOMES-UI-8B outperforms all evaluated end-to-end baselines across all four difficulty levels and three metrics. On Data-Scope-V1, it achieves 6.2% SR, 11.7% S_{prog}, and 15.7% S_{scope}, exceeding GUI-Owl-1.5-8B, the strongest end-to-end baseline on this level, by 3.1, 2.1, and 4.5 percentage points, respectively. These advantages persist as data-processing and filtering demands increase. On Data-Scope-V2 and Data-Scope-V3, DOMES-UI-8B exceeds the best end-to-end baseline scores in S_{prog} by 4.7 and 2.6 percentage points, and in S_{scope} by 5.6 and 3.6 percentage points, respectively. These results support better workflow progress and a better balance between the completeness and precision of data operations. Repeated evaluations on DC-V0 are reported in Appendix C.4.

Effectiveness of DOMES. To evaluate the effectiveness of DOMES, we compare it with history-based context and flat memory within the same agent framework in Table 6. We also compare against ASM and AgentProg, two agent frameworks that explicitly track task progress. All methods use the same planner and executor models. On AndroidWorld, DOMES achieves 86.5% SR, exceeding flat memory by 13.5 percentage points and ASM and AgentProg by 8.1 and 13.2 points, respectively. On DC-V0, it reaches 21.9%, compared with 15.6% for ASM and 12.5% for AgentProg. These results demonstrate the advantage of DOMES's unified representation in retaining values and tracking object-level progress.

Table 6: Framework-level comparisons using the same GPT-5.2 planner and UIIns executor under matched evaluation settings. Values are single-run success rates (%).

Mechanism	AW	DC-V0
Full visual and textual history	64.0	6.3
Three images+full text (Zhou et al., 2025)	73.9	9.4
Flat memory (Ye et al., 2025)	73.0	12.5
AgentProg (Tian et al., 2026)	73.3	12.5
ASM (Shi et al., 2026a)	**78.4**	**15.6**
DOMES (Ours)	**86.5**	**21.9**

4.3 Ablation Study

Contribution of Execution-State Information. To examine whether DOMES benefits from associating structured task data with processing states, we construct a status-free control from the same training trajectories. The control removes execution-status information from reasoning and memory while preserving data structure, values, constraints, and action supervision. As shown in Table 4, after SFT, DOMES improves SR over this control by 6.0 percentage points on AndroidWorld and 2.5 points on MobileWorld, and after STR-GRPO, by 8.6 and 5.2 points. To determine whether these improvements stem from longer supervised outputs, we additionally rewrite the status-free control to match DOMES's supervised token count. DOMES still outperforms this length-matched control at both training stages. These results demonstrate the value of explicit processing states beyond structured data retention and additional supervised tokens.

Contribution of STR-GRPO. Table 5 evaluates task success alongside the update rate, defined as the percentage of steps with nonempty DOMES outputs. Standard GRPO improves SR over SFT by 4.0 points on AndroidWorld and 3.5 points on MobileWorld, while reducing update rates by 2.5 and 4.9 points. STR-GRPO further reduces update rates by 13.8 and 14.6 points, reaching 31.9%

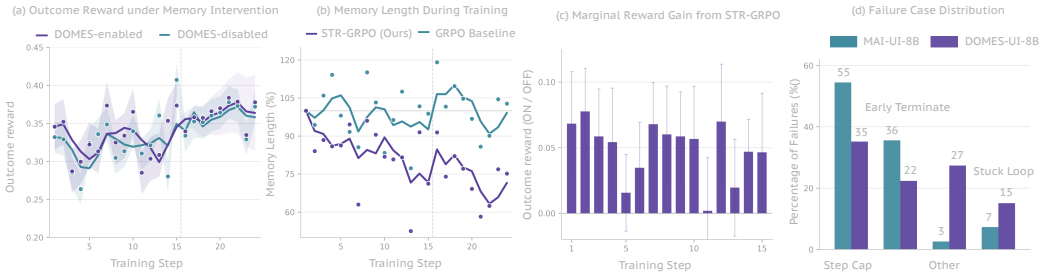

Figure 4: **STR-GRPO training dynamics and DataScope failure analysis. (a)** Task-completion rewards for DOMES-enabled and DOMES-disabled rollouts during STR-GRPO training. **(b)** DOMES output length under STR-GRPO and standard GRPO, normalized to 100% at the first training step. **(c)** Within-task reward differences between DOMES-enabled and DOMES-disabled rollouts. Positive values indicate higher rewards with DOMES. **(d)** Failure-type distributions for MAI-UI-8B and DOMES-UI-8B on DataScope. Percentages are computed among each agent's failed executions.

and 30.1%, respectively. Mean SR also increases from 74.7% to 76.7% on AndroidWorld and from 20.7% to 23.3% on MobileWorld. These results support the effectiveness of STR-GRPO in reducing state maintenance while improving mean task success.

4.4 Further Analysis

Learning Selective State Maintenance. Figure 4 (a) shows increasing task-completion rewards under both DOMES-enabled and DOMES-disabled conditions, with DOMES-enabled rollouts generally performing better. Since both conditions retain the ordinary interaction history, this advantage indicates that explicitly maintained task data and processing states provide useful information beyond history alone. In Figure 4 (b), DOMES output length decreases to approximately 70–75% of its initial value under STR-GRPO, compared with 90–100% under standard GRPO near the end of training. Together with the lower nonempty-update rates in Table 5, this suggests that STR-GRPO reduces both how often the agent generates state and how much it produces. Figure 4 (c) shows that the mean reward differences remain positive in later training, even as state output decreases. This sustained advantage suggests that less state generation can preserve the execution benefit of DOMES.

Failure Patterns on DataScope. To assess DOMES-UI's effectiveness, we manually inspect failed trajectories from DOMES-UI-8B and MAI-UI-8B on DataScope and compare their failure-type distributions in Figure 4 (d). Percentages are computed among each agent's failed executions. Step-limit exhaustion accounts for 35% of DOMES-UI-8B failures, compared with 55% for MAI-UI-8B. Together with its higher workflow-progress scores, this pattern is consistent with more effective execution within the interaction budget. Premature termination also accounts for a smaller share, at 22% versus 36%, consistent with explicit processing states helping the agent track unfinished work and inform completion decisions. However, stuck loops and other errors constitute larger shares of DOMES-UI-8B failures, at 15% and 27%, compared with 7% and 3% for MAI-UI-8B. The persistence of stuck loops indicates that explicit progress tracking alone does not prevent repeated unsuccessful actions. These results suggest improved workflow progress and completion tracking, while highlighting the need for better recovery when operations repeatedly fail.

5 Conclusion

We introduced DOMES, an object-centric execution-state representation that jointly maintains task data and processing states to support data retention and progress tracking in long-horizon mobile GUI tasks. Verified online trajectories teach DOMES-UI to construct and update this representation, while STR-GRPO reduces state maintenance by comparing execution with and without DOMES and accounting for its maintenance cost. We also introduced DataScope, a scalable online benchmark that evaluates workflow progress, target coverage, and scope control under increasing data-processing demands. Experiments on AndroidWorld, MobileWorld, and DataScope demonstrate strong execution performance and generalization to unseen environments, alongside reduced state-update rates. These findings highlight the value of coupling explicit execution states with learning when to maintain them.

AI USE STATEMENT

We used generative AI to produce task instructions, initialization data, final-state requirements, programmatic verifiers, and teacher demonstrations for training and evaluation. We also used AI assistance for manuscript drafting, language editing, and LaTeX formatting. Generated verifiers were checked against calibration cases and manually corrected when necessary, and teacher demonstrations were retained only after passing task-completion verification. We reviewed AI-assisted text and interpretations against the methods, experimental records, and cited sources. The authors take responsibility for the final content of this work, including all claims, data, and code.

ETHICS STATEMENT

Our experiments evaluate mobile GUI agents in virtual Android environments, and DataScope uses generated app data to construct controlled workflows. Since these agents can modify records across applications, incorrect actions may affect unrelated data or expose sensitive information in practical deployments. DataScope therefore evaluates both completion of required operations and modifications outside the intended task scope. The remaining failures indicate that deployment would require additional safeguards for user authorization, sensitive data handling, and recovery from unintended actions.

REPRODUCIBILITY STATEMENT

We will release our trained agents and the DataScope benchmark to support reproducibility. The main paper describes MOST, trajectory synthesis, STR-GRPO, and the evaluation protocols. Appendix E.1 provides the execution prompt, while Appendices A.4 and A.3 detail the training infrastructure and hyperparameters. Appendices B, B.3, and B.4 document benchmark construction, environment initialization, and verification procedures with a workflow example. Additional experiments in the appendix report repeated evaluations and confidence intervals to characterize variability.

REFERENCES

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhifang Guo, Qidong Huang, Jie Huang, Fei Huang, Binyuan Hui, Shutong Jiang, Zhaohai Li, Mingsheng Li, Mei Li, Kaixin Li, Zicheng Lin, Junyang Lin, Xuejing Liu, Jiawei Liu, Chenglong Liu, Yang Liu, Dayiheng Liu, Shixuan Liu, Dunjie Lu, Ruilin Luo, Chenxu Lv, Rui Men, Lingchen Meng, Xuancheng Ren, Xingzhang Ren, Sibo Song, Yuchong Sun, Jun Tang, Jianhong Tu, Jianqiang Wan, Peng Wang, Pengfei Wang, Qiuyue Wang, Yuxuan Wang, Tianbao Xie, Yiheng Xu, Haiyang Xu, Jin Xu, Zhibo Yang, Mingkun Yang, Jianxin Yang, An Yang, Bowen Yu, Fei Zhang, Hang Zhang, Xi Zhang, Bo Zheng, Humen Zhong, Jingren Zhou, Fan Zhou, Jing Zhou, Yuanzhi Zhu, and Ke Zhu. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*, 2025.

Yuxiang Chai, Hanhao Li, Jiayu Zhang, Liang Liu, Guangyi Liu, Guozhi Wang, Shuai Ren, Siyuan Huang, and Hongsheng Li. A3: Android agent arena for mobile gui agents. *arXiv preprint arXiv:2501.01149*, 2025.

Jingxuan Chen, Derek Yuen, Bin Xie, Yuhao Yang, Gongwei Chen, Zhihao Wu, Li Yixing, Xurui Zhou, Weiwen Liu, Shuai Wang, et al. Spa-bench: A comprehensive benchmark for smartphone agent evaluation. In *NeurIPS 2024 Workshop on Open-World Agents*, 2024.

Liangyu Chen, Hanzhang Zhou, Chenglin Cai, Jianan Zhang, Panrong Tong, Quyu Kong, Xu Zhang, Chen Liu, Yuqi Liu, Wenxuan Wang, et al. Ui-ins: Enhancing gui grounding with multi-perspective instruction-as-reasoning. *arXiv preprint arXiv:2510.20286*, 2025.

Kanzhi Cheng, Zehao Li, Zheng Ma, Nuo Chen, Jialin Cao, Qiushi Sun, Zichen Ding, Fangzhi Xu, Hang Yan, Jiajun Chen, et al. Openmobile: Building open mobile agents with task and trajectory synthesis. *arXiv preprint arXiv:2604.15093*, 2026.

DeepMind. Developing a computer use model. Google Blog, Oct 2025. URL `https://blog.google/technology/google-deepmind/gemini-computer-use-model/`. Accessed: October 22, 2025.

Zhangxuan Gu, Zhengwen Zeng, Zhenyu Xu, Xingran Zhou, Shuheng Shen, Yunfei Liu, Beitong Zhou, Changhua Meng, Tianyu Xia, Weizhi Chen, et al. Ui-venus technical report: Building high-performance ui agents with rft. *arXiv preprint arXiv:2508.10833*, 2025.

Quyu Kong, Xu Zhang, Zhenyu Yang, Nolan Gao, Chen Liu, Panrong Tong, Chenglin Cai, Hanzhang Zhou, Jianan Zhang, Liangyu Chen, et al. Mobileworld: Benchmarking autonomous mobile agents in agent-user interactive and mcp-augmented environments. *arXiv preprint arXiv:2512.19432*, 2025.

Sunjae Lee, Junyoung Choi, Jungjae Lee, Munim Hasan Wasi, Hojun Choi, Steve Ko, Sangeun Oh, and Insik Shin. Mobilegpt: Augmenting llm with human-like app memory for mobile task automation. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*, pp. 1119–1133, 2024.

Wei Li, William Bishop, Alice Li, Chris Rawles, Folawiyo Campbell-Ajala, Divya Tyamagundlu, and Oriana Riva. On the effects of data scale on ui control agents. *Advances in Neural Information Processing Systems*, 37:92130–92154, 2024a.

Yanda Li, Chi Zhang, Wenjia Jiang, Wanqi Yang, Bin Fu, Pei Cheng, Xin Chen, Ling Chen, and Yunchao Wei. Appagent v2: Advanced agent for flexible mobile interactions. *arXiv preprint arXiv:2408.11824*, 2024b.

Team OpenAI. Introducing openai o3 and o4-mini. *https://openai. com/index/introducing-o3-and-o4-mini/*, 2025.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, et al. Ui-tars: Pioneering automated gui interaction with native agents. *arXiv preprint arXiv:2501.12326*, 2025.

Ram Ramrakhya, Andrew Szot, Omar Attia, Bogdan Mazoure, Anh Nguyen, Yuhao Yang, Zhe Gan, Harsh Agrawal, and Alexander Toshev. Scaling synthetic task generation for agents via exploration. In *International Conference on Learning Representations*, volume 2026, pp. 123630–123660, 2026.

Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy Lillicrap. Androidinthewild: A large-scale dataset for android device control. *Advances in Neural Information Processing Systems*, 36:59708–59728, 2023.

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, et al. Androidworld: A dynamic benchmarking environment for autonomous agents. *arXiv preprint arXiv:2405.14573*, 2024.

Bytedance Seed. Seed1.8 model card: Towards generalized real-world agency. *arXiv preprint*, December 2025a. Technical Report.

ByteDance Seed. Ui-tars-1.5. `https://seed-tars.com/1.5`, 2025b.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*, 2024.

Yibo Shi, Jungang Li, Linghao Zhang, Zihao Dongfang, Biao Wu, Sicheng Tao, Yibo Yan, Chenxi Qin, Weiting Liu, Zhixin Lin, Hanqian Li, Yu Huang, Song Dai, Yonghua Hei, Yue Ding, Xiang Li, Shikang Wang, Chengdong Xu, Jingqi Liu, Xueying Ma, Zhiwen Zheng, Xiaofei Zhang, Bincheng Wang, Nichen Yang, Jie Wu, Lihua Tian, Chen Li, and Xuming Hu. Androtmem: From interaction trajectories to anchored memory in long-horizon gui agents, 2026a. URL `https://arxiv.org/abs/2603.18429`.

Yibo Shi, Jungang Li, Linghao Zhang, Zihao Dongfang, Biao Wu, Sicheng Tao, Yibo Yan, Chenxi Qin, Weiting Liu, Zhixin Lin, et al. Androtmem: From interaction trajectories to anchored memory in long-horizon gui agents. *arXiv preprint arXiv:2603.18429*, 2026b.

Qiushi Sun, Kanzhi Cheng, Zichen Ding, Chuanyang Jin, Yian Wang, Fangzhi Xu, Zhenyu Wu, Chengyou Jia, Liheng Chen, Zhoumianze Liu, et al. Os-genesis: Automating gui agent trajectory construction via reverse task synthesis. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5555–5579, 2025.

GELab Team. Gelab-zero: An advanced mobile agent inference system, 2025. URL https://github.com/stepfun-ai/gelab-zero.

Venus Team, Changlong Gao, Zhangxuan Gu, Yulin Liu, Xinyu Qiu, Shuheng Shen, Yue Wen, Tianyu Xia, Zhenyu Xu, Zhengwen Zeng, Beitong Zhou, Xingran Zhou, Weizhi Chen, Sunhao Dai, Jingya Dou, Yichen Gong, Yuan Guo, Zhenlin Guo, Feng Li, Qian Li, Jinzhen Lin, Yuqi Zhou, Linchao Zhu, Liang Chen, Zhenyu Guo, Changhua Meng, and Weiqiang Wang. Ui-venus-1.5 technical report, 2026a. URL https://arxiv.org/abs/2602.09082.

Venus Team, Changlong Gao, Zhangxuan Gu, Yulin Liu, Xinyu Qiu, Shuheng Shen, Yue Wen, Tianyu Xia, Zhenyu Xu, Zhengwen Zeng, et al. Ui-venus-1.5 technical report. *arXiv preprint arXiv:2602.09082*, 2026b.

Shizuo Tian, Hao Wen, Yuxuan Chen, Jiacheng Liu, Shanhui Zhao, Guohong Liu, Ju Ren, Yunxin Liu, and Yuanchun Li. Agentprog: Empowering long-horizon gui agents with program-guided context management. *arXiv preprint arXiv:2512.10371*, 2025.

Shizuo Tian, Hao Wen, Yuxuan Chen, Jiacheng Liu, Shanhui Zhao, Guohong Liu, Ju Ren, Yunxin Liu, and Yuanchun Li. Agentprog: Empowering long-horizon gui agents with program-guided context management, 2026. URL https://arxiv.org/abs/2512.10371.

Haoming Wang, Haoyang Zou, Huatong Song, Jiazhan Feng, Junjie Fang, Junting Lu, Longxiang Liu, Qinyu Luo, Shihao Liang, Shijue Huang, et al. Ui-tars-2 technical report: Advancing gui agent with multi-turn reinforcement learning. *arXiv preprint arXiv:2509.02544*, 2025a.

Zhenhailong Wang, Haiyang Xu, Junyang Wang, Xi Zhang, Ming Yan, Ji Zhang, Fei Huang, and Heng Ji. Mobile-agent-e: Self-evolving mobile assistant for complex tasks. *arXiv preprint arXiv:2501.11733*, 2025b.

Hao Wen, Yuanchun Li, Guohong Liu, Shanhui Zhao, Tao Yu, Toby Jia-Jun Li, Shiqi Jiang, Yunhao Liu, Yaqin Zhang, and Yunxin Liu. Autodroid: Llm-powered task automation in android. In *Proceedings of the 30th annual international conference on Mobile computing and networking*, pp. 543–557, 2024.

Han Xiao, Guozhi Wang, Hao Wang, Shilong Liu, Yuxiang Chai, Yue Pan, Yufeng Zhou, Xiaoxin Chen, Yafei Wen, and Hongsheng Li. Ui-mem: Self-evolving experience memory for online reinforcement learning in mobile gui agents. *arXiv preprint arXiv:2602.05832*, 2026.

Haiyang Xu, Xi Zhang, Haowei Liu, Junyang Wang, Zhaozai Zhu, Shengjie Zhou, Xuhao Hu, Feiyu Gao, Junjie Cao, Zihua Wang, et al. Mobile-agent-v3. 5: Multi-platform fundamental gui agents. *arXiv preprint arXiv:2602.16855*, 2026.

Yifan Xu, Xiao Liu, Xueqiao Sun, Siyi Cheng, Hao Yu, Hanyu Lai, Shudan Zhang, Dan Zhang, Jie Tang, and Yuxiao Dong. Androidlab: Training and systematic benchmarking of android autonomous agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2144–2166, 2025.

Haolong Yan, Jia Wang, Xin Huang, Yeqing Shen, Ziyang Meng, Zhimin Fan, Kaijun Tan, Jin Gao, Lieyu Shi, Mi Yang, Shiliang Yang, Zhirui Wang, Brian Li, Kang An, Chenyang Li, Lei Lei, Mengmeng Duan, Danxun Liang, Guodong Liu, Hang Cheng, Hao Wu, Jie Dong, Junhao Huang, Mei Chen, Renjie Yu, Shunshan Li, Xu Zhou, Yiting Dai, Yineng Deng, Yingdan Liang, Zelin Chen, Wen Sun, Chengxu Yan, Chunqin Xu, Dong Li, Fengqiong Xiao, Guanghao Fan, Guopeng Li, Guozhen Peng, Hongbing Li, Hang Li, Hongming Chen, Jingjing Xie, Jianyong Li, Jingyang Zhang, Jiaju Ren, Jiayu Yuan, Jianpeng Yin, Kai Cao, Liang Zhao, Liguo Tan, Liying Shi, Mengqiang Ren, Min Xu, Manjiao Liu, Mao Luo, Mingxin Wan, Na Wang, Nan Wu, Ning Wang, Peiyao Ma, Qingzhou Zhang, Qiao Wang, Qinlin Zeng, Qiong Gao, Qiongyao Li, Shangwu Zhong, Shuli Gao, Shaofan Liu, Shisi Gao, Shuang Luo, Xingbin Liu, Xiaojia Liu, Xiaojie Hou, Xin Liu,

Xuanti Feng, Xuedan Cai, Xuan Wen, Xianwei Zhu, Xin Liang, Xin Liu, Xin Zhou, Yingxiu Zhao, Yukang Shi, Yunfang Xu, Yuqing Zeng, Yixun Zhang, Zejia Weng, Zhonghao Yan, Zhiguo Huang, Zhuoyu Wang, Zheng Ge, Jing Li, Yibo Zhu, Binxing Jiao, Xiangyu Zhang, and Daxin Jiang. Step-gui technical report, 2025. URL https://arxiv.org/abs/2512.15431.

Leyang Yang, Ziwei Wang, Xiaoxuan Tang, Sheng Zhou, Dajun Chen, Wei Jiang, and Yong Li. Probench: Benchmarking gui agents with accurate process information. *arXiv preprint arXiv:2511.09157*, 2025.

Jiabo Ye, Xi Zhang, Haiyang Xu, Haowei Liu, Junyang Wang, Zhaoqing Zhu, Ziwei Zheng, Feiyu Gao, Junjie Cao, Zhengxi Lu, et al. Mobile-agent-v3: Fundamental agents for gui automation. *arXiv preprint arXiv:2508.15144*, 2025.

Chi Zhang, Zhao Yang, Jiaxuan Liu, Yanda Li, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. Appagent: Multimodal agents as smartphone users. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, pp. 1–20, 2025.

Yujie Zheng, Zikang Liu, Xin Zhao, and Ji-Rong Wen. A task-state representation for long-horizon mobile gui agents, 2026. URL https://arxiv.org/abs/2607.00502.

Hanzhang Zhou, Xu Zhang, Panrong Tong, Jianan Zhang, Liangyu Chen, Quyu Kong, Chenglin Cai, Chen Liu, Yue Wang, Jingren Zhou, et al. Mai-ui technical report: Real-world centric foundation gui agents. *arXiv preprint arXiv:2512.22047*, 2025.

Appendix Contents

A Implementation and Training Details 14
- A.1 Training Task Construction 14
- A.2 STR-GRPO Training Details 15
- A.3 Training Resources and Configuration 15
- A.4 Online RL Environment and Infrastructure 15
- A.5 Result Sources and Evaluation Settings 15
- A.6 Interaction Budgets 16
- A.7 Framework Baseline Implementations 16
- A.8 Construction of Representation Controls 16

B DataScope Construction and Verification 17
- B.1 Task Families and Difficulty Control 17
- B.2 Human Validation 17
- B.3 Environment Initialization and Verifier Calibration 17
- B.4 Workflow Template Example 17
- B.5 Task-Family Statistics 17

C More Experiments 19
- C.1 Effect of Teacher Quality 19
- C.2 Ablations of the State-Update Penalty 20
- C.3 Statistical Comparison of GRPO and STR-GRPO 20
- C.4 Performance across Training Seeds 20
- C.5 Inference Overhead of DOMES 21
- C.6 Schema Errors and Correction 21

D Qualitative Analysis 21
- D.1 Flat Memory versus DOMES 21
- D.2 Data Retention and Progress Tracking across Apps 23
- D.3 Repeated Actions during Recipient Entry 23

E Execution and Task-Generation Prompts 23
- E.1 Agent Execution Prompt 23
- E.2 Benchmark Task-Generation Prompt 24

A Implementation and Training Details

A.1 Training Task Construction

We construct 120 manually designed and verified training templates in a virtual Android environment with 20 applications. Each template specifies the task goal, task data, initialization procedure, and terminal-state requirements. An LLM-assisted generator (OpenAI, 2025) produces approximately 1.2K candidate instances, of which approximately 1.1K remain executable after filtering malformed, invalid, or unreachable instances. The training instances are disjoint from the 128 DataScope evaluation instances described in Appendix B.

Teacher rollouts use GPT-5.2 (OpenAI, 2025) as the planner and UIIns (Chen et al., 2025) as the executor. At each interaction step, the teacher produces reasoning, one GUI action, and an optional structured state update. The teacher uses DOMES when information must be retained or compared across steps and otherwise emits an empty state block. A rollout is retained only when its final environment state passes the task-specific verifier. This procedure yields 21,713 step-level SFT samples. SFT supervises the reasoning, action, and state tokens in each response. Every setting contains exactly one supervised action per interaction step. Final-state verification establishes task completion but does not certify the correctness of every intermediate reasoning statement or state update. Schema reliability is evaluated separately in Appendix C.6.

A.2 STR-GRPO Training Details

For each task initialization, STR-GRPO collects a group of 16 rollouts under the current policy, with eight assigned to DOMES-enabled execution and eight to DOMES-disabled execution. Enabled rollouts propagate the explicit state across turns. Disabled rollouts mask this state channel while retaining the task instruction, current observation, and ordinary action history.

The trajectory reward combines binary terminal success with a penalty on the fraction of steps containing nonempty state updates. The penalty applies only to DOMES-enabled rollouts, with the default coefficient set to $\alpha = 0.05$. Advantages are normalized over the combined enabled and disabled rollouts for the same task. The policy loss covers reasoning, action, and state tokens for enabled rollouts, while state tokens are masked from the loss for disabled rollouts. This provides a task-controlled learning signal for state use, rather than a separate causal attribution for each update. In the ablation experiments, GRPO and STR-GRPO start from the same SFT checkpoint and share RL tasks, rollout budgets, optimization steps, hardware resources, and evaluation settings.

A.3 Training Resources and Configuration

Table 7 summarizes the reported main training configuration. SFT uses 128 NVIDIA H20 GPUs across 16 nodes and takes approximately two hours. The vision encoder is frozen during SFT, and training uses bfloat16 precision, FlashAttention, gradient checkpointing, Liger kernels, and DeepSpeed ZeRO-2 offload. Online RL initializes the actor from the SFT checkpoint and uses 256 NVIDIA H20 GPUs across 32 nodes for approximately one day. Each training node has 96 CPU cores and approximately 900 GB RAM.

The environment service runs on three bare-metal servers, each with 96 CPU cores and 384 GB RAM. Of 200 environment containers, 128 are active during rollout collection and the remainder provide backup capacity. During RL training, the cumulative response length is capped at 81,920 tokens per rollout. End-to-end evaluation instead limits each interaction step to 2,048 newly generated tokens.

A.4 Online RL Environment and Infrastructure

Long mobile rollouts vary substantially in duration and contain many screenshots and interaction turns. Following MAI-UI (Zhou et al., 2025), we use a `verl`-based training stack with HybridFlow (Sheng et al., 2024) to collect interactions asynchronously and optimize long multimodal trajectories.

Rollout Collection. An agent loop dispatches requests to inference servers and manages environment sessions asynchronously. Request load balancing and prefill caching support inference over long histories. Backup environment containers replace failed sessions. Each rollout batch is associated with the policy version that generated it and retains its behavior log probabilities for optimization. The reported configuration disables rollout replay.

Long-context Optimization. Tensor and context parallelism partition model computation and trajectory contexts across GPUs. The reported configuration uses tensor parallelism of degree 2, context parallelism of degree 2, and pipeline parallelism of degree 1. The rollout engine uses a GPU memory utilization ratio of 0.55. Screenshots are resized for RL to reduce visual input size, with the reported resolution given in Table 7.

A.5 Result Sources and Evaluation Settings

AndroidWorld. Baseline scores in Table 1 are taken from the corresponding papers, technical reports, or official evaluations. We evaluate DOMES-UI using the public AndroidWorld tasks, official success verifiers, and task-specific step limits. Published reference scores were obtained under their original evaluation settings and do not constitute a unified resource-controlled comparison.

MobileWorld. Baseline scores in Table 2 are taken from the official MobileWorld report and website. To assess generalization, we compare only agents trained without exposure to the MobileWorld evaluation environments. DOMES-UI uses the public evaluation harness and the same maximum of 50 interaction steps per task.

DataScope. We reproduce end-to-end baselines using their official checkpoints, prompts, action parsers, and coordinate conventions, with interface adaptations for the evaluation harness. These agents run through vLLM in the same serving environment with eight NVIDIA H20 GPUs. They share task initializations, task-specific step limits, termination conditions, and programmatic verifiers. No additional wall-clock timeout is imposed.

Table 7: Training resources and configuration. Training sequence and response limits are separate from evaluation generation limits.

Configuration	SFT	Online RL
Base model	Qwen3-VL-4B/8B-Instruct	SFT checkpoint
Training GPUs	128 NVIDIA H20	256 NVIDIA H20
Node setup	16 nodes, 8 GPUs/node	32 nodes, 8 GPUs/node
CPU / RAM per node	96 cores / ∼900 GB	96 cores / ∼900 GB
Training time	∼2 hours	∼1 days
Precision	bfloat16	bfloat16
Max sequence / response length	20,768	81,920
Vision encoder	Frozen	–
Parallelism	ZeRO-2 offload	TP=2, PP=1, CP=2
Batch size	1 per GPU	global 32, mini-batch 32
Rollout group size	–	16
Image limit / resolution	max pixels 12,845,056	1600×720
Environment servers	–	3 bare-metal servers
Environment server configuration	–	96 cores / 384 GB each
Environment containers	–	200 total, 128 active

Decoding. DOMES-UI-8B and the end-to-end agents reproduced on DataScope use greedy decoding with temperature 0, top-p 1.0, top-k disabled, and at most 2,048 newly generated tokens per interaction step. The evaluated outputs were checked for truncation under this cap. Framework experiments use GPT-5.2 with temperature 0 and at most 16,384 output tokens per planner call. UIIns uses temperature 0 and at most 512 output tokens per executor call.

A.6 Interaction Budgets

AndroidWorld follows its official task-specific limits, and MobileWorld uses a 50-step limit. For DataScope, we estimate the interaction budget as:

$$\hat{H} = c_0 + c_e N_{\text{target}} + c_s N_{\text{scan}} + c_a (N_{\text{app}} - 1), \qquad H = \left\lceil 2\hat{H} \right\rceil . \tag{8}$$

Here N_{target} denotes the number of entries that require creation, modification, deletion, or transfer. N_{scan} denotes the number of source entries that must be inspected or filtered, and N_{app} denotes the number of participating apps. We set $c_0 = 10$, $c_e = 6$, $c_s = 0.5$, and $c_a = 2$ to approximate the interaction costs of initial navigation, target operations, source inspection, and cross-app transitions, respectively. The final step limit H doubles this estimate and rounds up to an integer. All human reference trajectories complete within their assigned limits. Budgets are fixed before evaluation and applied uniformly to all methods on each task.

A.7 Framework Baseline Implementations

We distinguish comparisons within our agent framework from comparisons with other agent frameworks. The full-history, recent-image-history, flat-memory, and DOMES variants use the same host framework and differ in how they retain execution context. Full history retains visual and textual interaction history. The recent-image variant retains the three latest screenshots and the full textual history. Flat memory records key task information following Mobile-Agent-V3 (Wang et al., 2025b). DOMES associates task data with explicit processing states.

ASM (Shi et al., 2026a) and AgentProg (Tian et al., 2026) are reproduced as agent frameworks while retaining their respective execution workflows and state-management mechanisms. We replace their planner and executor models with GPT-5.2 and UIIns to match the underlying models used by DOMES. All framework methods share action spaces, decoding settings, task budgets, and evaluation criteria. This controls model capability and evaluation conditions while preserving framework-specific behavior. The framework results evaluate DOMES as a state representation and are separate from the trained DOMES-UI model results.

A.8 Construction of Representation Controls

Status-free Control. We construct this control from the same successful GPT-5.2 trajectories used for DOMES. GPT-5.2 rewrites only the reasoning and state blocks to remove execution-status fields, status-based grouping, and descriptions of completed, remaining, or skipped work. Task instructions, observations, action sequences, data structure, values, constraints, and other non-status information

are preserved. Rule-based checks reject rewritten samples containing invalid values, new task content, or residual status information. Every sample retains the original supervised action.

Length-matched Control. We additionally rewrite the status-free reasoning and state blocks to match the number of loss-bearing tokens in the corresponding DOMES sample. Rewriting uses only the original task objects and constraints and adds no new facts, actions, or planning information. Token counts are checked with the backbone tokenizer, and samples differing by more than five tokens are regenerated. All representations use the same tasks, observations, action sequences, data volume, backbone, and training budget. Their STR-GRPO variants also share the RL tasks, rollout budget, optimization steps, state intervention, and maintenance-cost objective.

B DataScope Construction and Verification

B.1 Task Families and Difficulty Control

DataScope contains 32 task families defined by manually designed templates, each instantiated at four difficulty levels, yielding 128 evaluation instances across 14 mobile apps. All task templates involve multiple apps. Each template specifies the task goal, participating apps, data fields, object relationships, initialization examples, and required outputs. Difficulty is controlled by varying the number of target entries and same-schema distractors while preserving the workflow structure and operation types. Processing more entries may require repeating the same operations and therefore more GUI actions.

Distractors share the schema or data type of target entries but do not satisfy the task's constraints. Both targets and distractors are initialized in app storage and accessed through the GUI. DataScope therefore evaluates whether agents can identify and process all target entries without modifying unrelated data. Table 8 reports the data counts and participating apps for each task family.

B.2 Human Validation

Each evaluation instance is independently reviewed by three annotators. We retain an instance only when all three agree that its instruction is clear and consistent with the initialized environment, the task is executable, and completion can be verified programmatically. Four of the 128 instances required minor corrections before inclusion. The parameterized templates support generating additional instances beyond the current evaluation set.

B.3 Environment Initialization and Verifier Calibration

Following AndroidWorld (Rawles et al., 2024), we initialize tasks by writing target entries and distractors into app-specific storage, including SQLite databases and files. The expected outputs define the final state, including created or updated records and distractors that must remain unchanged.

An LLM generates a task-specific verifier from the initializer, final-state requirements, and calibration cases. Each calibration case specifies a controlled environment state, an expected binary task-success result, and expected S_{prog} and S_{scope} scores. The cases cover successful completion, omitted operations, distractor modifications, incomplete app-level subgoals, and incorrect field values. We load each state into the environment to simulate an execution result and compare the verifier's scores with the expected values. When a verifier fails a calibration case, we summarize the failed checks and discrepancies between its scores and the expected values. The generator uses this feedback to regenerate the verifier, which is retested until all calibration cases pass.

B.4 Workflow Template Example

Listing 1 presents a meeting-preparation template spanning Calendar, Tasks, Contacts, Markor, and Messages. It specifies cross-app data relationships, difficulty settings, initialization data, and expected outputs.

B.5 Task-Family Statistics

Each subset cell in Table 8 reports T/N/O: task-relevant entries, same-schema distractors, and required data operations, respectively. Entry counts follow the counting rules for each task family, including its treatment of associated records. An operation is counted at the task-defined data-operation level, rather than as an individual GUI interaction. Multiple entries may contribute to one operation, and one entry may require multiple operations, so T and O need not be equal. The reported subsets have increasing average workloads; individual task-family counts need not increase at every level.

Table 8: **Task-family details across benchmark subsets.** Each subset cell reports **T/N/O**, where **T**, **N**, and **O** denote the numbers of target entries, same-schema distractors, and required operations, respectively. Applications are shown as color-coded labels for readability.

Task Family	Apps	#Apps	DC-V0	DC-V1	DC-V2	DC-V3
CalendarFocusTracksRetro Playlist	Files, Music, Markor, Calendar, Tasks	3	2/2/3	3/7/3	5/13/3	6/20/3
CalendarMeetingPrepAnd Notify	Contacts, Markor, Messages, Calendar	5	4/5/3	5/12/3	20/43/12	36/100/22
CalendarTodayAlarmsInClock	Clock, Markor	3	1/3/2	2/6/3	3/10/4	4/17/5
CallLogContactTodoMarkor Sync	Contacts, Markor, Phone, Tasks	4	2/0/2	3/0/4	9/4/4	10/2/10
CallLogTopContactsFavorite	Contacts, Phone	2	5/1/3	5/3/3	11/10/3	13/25/3
ChromeInviteCodeAndSms Send	Chrome, Messages	2	6/1/5	8/3/6	7/7/5	16/18/9
ExpensePurgeHousingFoodLog Markor	Markor, Expense Book	2	2/1/3	2/2/3	4/1/3	6/3/3
FilesClosedProjectContact Delete	Contacts, Files	2	2/5/1	2/7/1	9/14/3	15/36/3
FilesDatedReimburseTxtTo Expense	Files, Expense Book	2	1/2/1	2/3/3	3/5/6	4/10/8
FilesPartyMenuRecipesTo Broccoli	Broccoli, Files, Markor	3	2/1/2	2/3/2	2/6/2	2/10/2
FilesSetlistExportRetroM3u	Files, Music, Calendar	2	4/1/4	6/2/4	8/6/4	12/9/4
FilesWorkEventsToCalendar	Files, Messages	3	2/2/2	3/6/4	6/10/8	8/17/13
FilesWorkMergeToMarkorCsv	Files, Markor	2	1/1/1	2/1/1	4/3/1	6/8/1
GalleryImageListToMarkor	Gallery, Markor	2	1/1/1	2/3/1	4/6/3	6/12/4
GalleryVlcArtistPlaylistsFrom Images	Gallery, VLC, Markor	2	3/2/2	8/4/4	12/12/6	20/21/8
MarkorTaskAssignment Distribute	Contacts, Messages, Tasks, Broccoli, Markor	4	1/2/2	2/4/8	4/35/20	5/72/33
RecipeNutFreePurgeLogMarkor	Files, Messages, Markor	4	4/6/3	6/4/3	7/12/3	9/14/3
RetroLightMusicPartyMarkor Sms	Messages, Music, Markor	3	5/2/3	8/3/3	12/6/3	19/8/4
SmsAgentPhoneToMarkorTxt	Markor, Messages, Markor	2	1/2/2	2/4/2	4/7/2	6/30/2
SmsAlbumQueueRetroMarkor	Messages, Music, Messages	3	3/4/2	5/6/2	7/13/2	8/21/2
SmsColleagueTripReimburseTo Expense	Messages, Expense Book	2	2/4/4	2/5/4	2/6/5	4/13/7

Continued on next page

Table 9: Teacher comparison using approximately 0.36K matched trajectories from tasks completed by both teachers, with equal training budgets. AW and MW denote AndroidWorld and MobileWorld. Values are success rates (%). Green/purple marks the best/second-best results.

	SFT		+ STR-GRPO	
Teacher	**AW**	**MW**	**AW**	**MW**
Qwen3-VL-8B				
GPT-4.1	**63.8**	**14.7**	**68.1**	**17.2**
GPT-5.2	**65.5**	**15.5**	**69.8**	**18.1**

Table 10: Sensitivity to maintenance cost α with other settings fixed. SR is the success rate, and Update is the percentage of steps with nonempty DOMES outputs. All values are percentages. Green/purple marks the best/second-best success rate.

	AndroidWorld		MobileWorld	
α	**SR**	**Update**	**SR**	**Update**
0.025	73.3	38.4	20.7	36.5
0.05 (default)	**76.7**	31.9	**23.3**	30.1
0.10	**74.1**	25.6	**21.6**	24.8

Task Family	**Apps**	**#Apps**	**DC-V0**	**DC-V1**	**DC-V2**	**DC-V3**
SmsDavidBugfixToCalendar	Calendar, Messages	2	1/1/1	2/4/2	4/8/4	6/10/6
SmsDavidClientsToContacts	Contacts, Messages	2	1/2/1	1/4/2	3/18/7	4/48/10
SmsDavidTodoToTasks	Messages, Tasks	2	1/2/2	1/4/3	2/15/7	3/40/12
SmsExpenseAuditAndReport	Messages, Expense Book, Markor	3	1/5/2	3/8/2	5/16/2	6/31/2
SmsFamilyRecipeToBroccoli	Broccoli, Messages, Markor	3	1/3/3	2/5/4	3/9/5	4/13/6
SmsFriendRecommendations VlcPlaylists	Messages, VLC, Broccoli	2	2/3/2	5/5/3	13/13/9	19/20/14
SmsPotluckRecipeBroccoli Calendar	Calendar, Messages, Files	4	1/2/3	2/5/3	4/6/3	5/9/3
SmsStockDataToMarkorCsv	Markor, Messages	2	1/2/1	2/6/1	4/12/1	7/27/1
SmsWorkPlanToMarkorTodo	Markor, Messages	2	1/2/1	1/4/1	3/10/1	5/25/1
SmsWorkoutPlaylistRetro Markor	Markor, Messages, Music	3	3/2/3	5/7/3	5/9/3	6/20/3
VlcPlaylistsToMarkorMdFiles	VLC, Messages, Contacts	3	6/3/4	8/4/4	8/11/4	14/17/4

C More Experiments

The following studies supplement the main ablations by examining teacher quality, variation across runs, and the cost and accuracy of state maintenance.

C.1 Effect of Teacher Quality

To evaluate the effect of the teacher model, we use GPT-4.1 and GPT-5.2 separately to generate demonstrations and retain only tasks that both teachers complete successfully. This yields two training sets covering the same tasks, each containing 0.36K trajectories from one teacher. We train Qwen3-VL-8B separately on each set using the same training budget and hardware configuration.

As shown in Table 9, models trained on GPT-5.2 demonstrations achieve 1.7 percentage points higher SR on AndroidWorld after both SFT and STR-GRPO. The corresponding differences on MobileWorld are 0.8 and 0.9 points. STR-GRPO further improves SR with either teacher, indicating that its benefits are not limited to GPT-5.2 demonstrations. This comparison controls task selection but does not assess differences in the range of tasks each teacher can complete.

Table 12: Inference overhead from paired replay with DOMES-UI-8B on AndroidWorld steps carrying DOMES. Values are per-step means. Latency includes DOMES generation.

Metric	W/O	W/	Δ
Input tokens	9,425	10,457	+1,032
Time to first token (s)	3.87	4.24	+0.37
DOMES output tokens	0	236	+236
Inference latency (s)	5.44	12.05	+6.61

Input tokens and time to first token increase by 11.0% and 9.5%, respectively. The median latency difference is 2.1 s. Replay uses batch size 1, greedy decoding, 24 CPU cores, 240 GB RAM, and two NVIDIA H20 GPUs.

Table 13: Schema errors and recovery in tasks using DOMES. Error prevalence is measured over these tasks, and recovery rates over tasks with detected errors. Item-value errors are excluded. Values are percentages.

Metric	AW	DataScope
Tasks with schema errors	3.0	12.3
Among tasks with schema errors		
All errors subsequently corrected	100.0	75.0
Errors remaining at task end	0.0	25.0

AW denotes AndroidWorld.

C.2 Ablations of the State-Update Penalty

We evaluate $\alpha \in \{0.025, 0.05, 0.10\}$ while keeping all other training and evaluation settings fixed. The coefficient α controls the penalty on the fraction of steps containing nonempty DOMES updates.

Table 10 suggests that the default $\alpha = 0.05$ achieves the highest SR on both benchmarks. Compared with this setting, reducing α to 0.025 increases update rates by 6.5 and 6.4 percentage points on AndroidWorld and MobileWorld, respectively, while lowering SR by 3.4 and 2.6 points. Increasing α to 0.10 lowers update rates by 6.3 and 5.3 points, but also lowers SR by 2.5 and 1.7 points. Thus, more frequent updates do not necessarily improve execution, while a stronger penalty can reduce task success. Among the tested settings, $\alpha = 0.05$ provides the best task performance with fewer updates than $\alpha = 0.025$.

C.3 Statistical Comparison of GRPO and STR-GRPO

We train GRPO and STR-GRPO using three matched random seeds and report the mean and standard deviation in Table 5. To assess uncertainty in their differences, we compute 95% confidence intervals using 10,000 paired bootstrap resamples over evaluation tasks. Each resample uses the same sampled tasks for both methods.

Table 11: STR-GRPO minus GRPO in percentage points. The 95% confidence intervals use 10,000 paired bootstrap resamples over evaluation tasks.

Metric	Δ	95% CI
AndroidWorld		
Success rate	+2.0	[−1.7, +5.8]
Update rate	−13.8	[−16.3, −11.4]
MobileWorld		
Success rate	+2.6	[−1.2, +6.5]
Update rate	−14.6	[−17.1, −12.1]

As shown in Table 11, STR-GRPO reduces update rates by 13.8 percentage points on AndroidWorld and 14.6 points on MobileWorld. Both confidence intervals for these differences lie below zero, supporting a consistent reduction in state updates. Mean SR is also higher by 2.0 and 2.6 points, respectively, although both confidence intervals include zero. These results provide statistical evidence that STR-GRPO reduces state maintenance, while its higher mean SR does not establish a statistically significant improvement in task success.

C.4 Performance across Training Seeds

On AndroidWorld and MobileWorld. We train DOMES-UI-8B with three independent random seeds and evaluate each resulting model on AndroidWorld and MobileWorld. For comparison, we also train MAI-UI-8B with three seeds and evaluate it on AndroidWorld. Table 14 reports mean SR and standard deviation across the three training runs. DOMES-UI-8B achieves $76.7 \pm 1.7\%$ SR on AndroidWorld, compared with $70.7 \pm 0.9\%$ for MAI-UI-8B, and $23.3 \pm 0.9\%$ on MobileWorld. The AndroidWorld comparison shows a 6.0 percentage point advantage in mean SR for DOMES-UI-8B across the evaluated training seeds.

On DataScope. We evaluate DOMES-UI-8B and MAI-UI-8B on DC-V0 using models from three independent training runs. For the framework comparison, we perform three evaluation runs per method using the same GPT-5.2 planner and UIIns executor. Table 15 reports mean SR and standard

deviation separately for the two settings. DOMES-UI-8B achieves a mean SR of 11.5%, compared with 4.2% for MAI-UI-8B. Among the frameworks, DOMES achieves $19.8 \pm 1.8\%$, compared with $14.6 \pm 4.8\%$ for ASM and $11.5 \pm 4.8\%$ for AgentProg. DOMES therefore has both a higher mean SR and lower variation across the three framework evaluation runs. These results extend the single-run comparisons by showing higher average task success in both experimental settings.

C.5 Inference Overhead of DOMES

To measure the additional inference cost of DOMES, we select AndroidWorld steps with a non-empty DOMES state and compare inference with and without DOMES using the same DOMES-UI-8B checkpoint. For each selected step, the task instruction, screenshot, and interaction history are identical in both settings. With DOMES enabled, the model receives the stored state and can generate a state update. With DOMES disabled, the stored state is removed, and the model cannot generate a state update. Both settings use batch size 1, greedy decoding, 24 CPU cores, 240 GB RAM, and two NVIDIA H20 GPUs.

As shown in Table 12, enabling DOMES adds an average of 1,032 input tokens, an 11.0% increase, and increases time to first token by 0.37 s, or 9.5%. Including state generation, mean inference latency per step increases from 5.44 s to 12.05 s. The latency difference between the enabled and disabled settings has a mean of 6.61 s and a median of 2.1 s across the selected steps. These measurements capture the additional cost on steps using DOMES and motivate reducing unnecessary state updates. They do not represent average latency over all interaction steps or total task execution time.

Table 14: Success rates (%) are reported as the mean $\pm$ standard deviation over three independent training runs.

Benchmark	Agent	SR
AndroidWorld	MAI-UI-8B	70.7 ± 0.9
AndroidWorld	**DOMES-UI-8B**	$\mathbf{76.7 \pm 1.7}$
MobileWorld	**DOMES-UI-8B**	23.3 ± 0.9

C.6 Schema Errors and Correction

We analyze evaluation tasks in which the agent uses DOMES. A task is counted as containing a schema error if any DOMES state omits a required field or incorrectly defines a field, constraint, or mapping between objects and fields. Missing fields are judged against the information available in the task instruction and observations at that step. Errors in stored item values are excluded.

For AndroidWorld, GPT-5.2 derives reference schemas specifying the required objects, fields, constraints, and mappings from task instructions and trajectory screenshots. We manually review all tasks with detected errors and 30 randomly selected tasks classified as error-free. For DataScope, reference schemas are generated directly from the task templates. An error-containing task is classified as corrected if all detected errors are resolved in subsequent states and none remain at task completion.

Table 13 reports schema errors in 3.0% of AndroidWorld tasks using DOMES and 12.3% of DataScope tasks using DOMES. All detected AndroidWorld errors are subsequently corrected. On DataScope, all detected errors are corrected in 75.0% of affected tasks, while at least one error persists in the remaining 25.0%. These results indicate that schema errors can be corrected during execution, although correction alone does not imply accurate item values or successful task completion.

Table 15: Repeated evaluation on DC-V0 using three training seeds for end-to-end agents and three evaluation runs for frameworks. SR is reported as mean $\pm$ standard deviation (%). Green/purple marks the best/second-best mean SR within each group.

Method	SR
End-to-end agents	
Three independent training runs	
MAI-UI-8B	4.2 ± 1.8
DOMES-UI-8B	11.5 ± 4.8
Frameworks with GPT-5.2 + UIIns	
Three evaluation runs	
AgentProg	11.5 ± 4.8
ASM	14.6 ± 4.8
DOMES	19.8 ± 1.8

D Qualitative Analysis

D.1 Flat Memory versus DOMES

Figure 5 compares flat memory and DOMES within the same agent framework on a task requiring deletion of three specified expense entries while preserving same-schema distractors. The flat-memory agent records target names but does not explicitly track completed deletions. It repeatedly searches for entries it has already processed and reaches the 100-step limit.

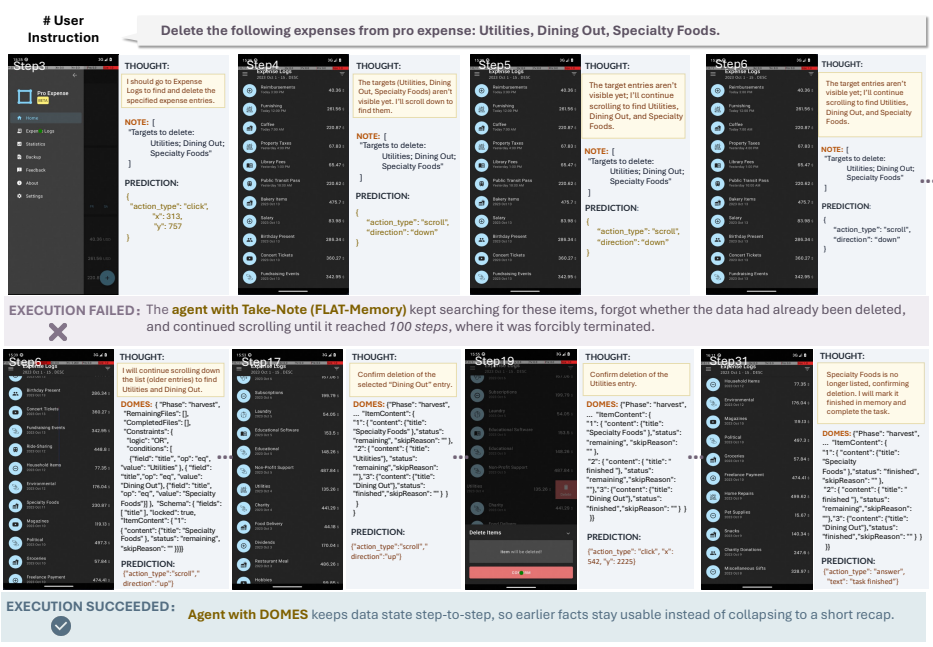

Figure 5: Comparison of flat memory and DOMES on an expense-deletion task. The flat-memory agent reaches the 100-step limit, while the DOMES agent tracks completed deletions and finishes in 31 steps. Some state content is omitted for readability.

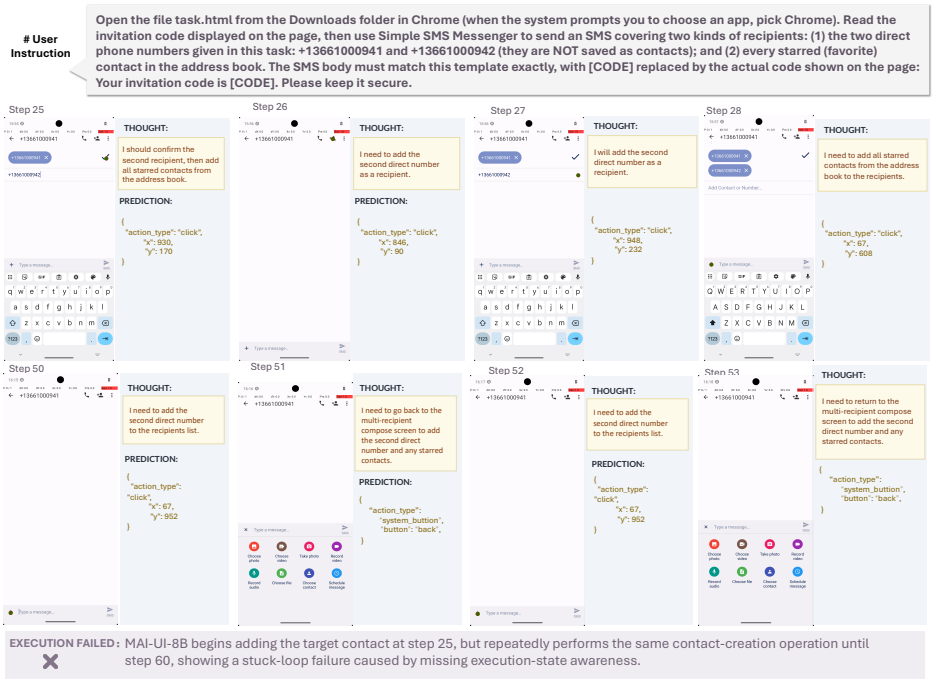

Figure 6: Repeated recipient-entry attempts by MAI-UI-8B on DataScope. The agent has not confirmed the second recipient by step 60 and has yet to process the starred contacts.

With DOMES, each target is initially marked `remaining` and updated to `finished` after deletion. This record identifies which entries still require action, and the agent completes the task in 31 steps. The example illustrates the benefit of tracking processing states alongside target information.



Figure 7: A recipe-transfer trajectory on AndroidWorld. DOMES stores recipe fields collected from Simple Gallery Pro and tracks which recipes have been saved in Broccoli.

D.2 Data Retention and Progress Tracking across Apps

Figure 7 shows DOMES-UI transferring three recipes from Simple Gallery Pro to Broccoli in 58 steps. The agent first collects the recipe fields into structured items, each marked `remaining`. It then uses the stored values to enter recipes into Broccoli and marks each recipe `finished` after saving it. This trajectory illustrates how DOMES preserves information across app transitions while tracking progress through multiple target items.

D.3 Repeated Actions during Recipient Entry

Figure 6 shows a DataScope task requiring an SMS to two specified phone numbers and all starred contacts, with message content taken from a local HTML file. MAI-UI-8B adds the first recipient but repeatedly attempts to add the second, cycling between the recipient field, back navigation, and the compose screen. By step 60, the second recipient remains unconfirmed and the starred contacts have not been processed. This example shows how repeated attempts at one operation can prevent progress on the remaining task. It highlights the need to verify whether an action succeeded and recover when repeated attempts fail.

E Execution and Task-Generation Prompts

E.1 Agent Execution Prompt

The execution prompt specifies the action space, response format, and rules for updating DOMES (*i.e.*, the memory block in the prompt). Each response contains a reasoning block, exactly one executable tool call, and a memory block containing either structured state or an empty JSON object.

Prompt 1. DOMES-Augmented Agent Execution Prompt

Task. You are a GUI agent. You are given a task goal, previous action history, and screenshots of the current screen. You need to perform the **NEXT** action to complete the task.

Output Format. For each step, output exactly **THREE** blocks in this order:

```
<thinking>
... return the thinking process
</thinking>
<tool_call>
{"name": "mobile_use", "arguments": <args-json-object>}
</tool_call>
<memory>
{ ...valid JSON dict or {} ... }
</memory>
```

Rules.
- Output exactly **one** tool call per step.

Action Space. Choose exactly **one** action per step:

Action	JSON Format
`open_app`	`{"action": "open_app", "app_name": "<Available Apps>"}`
`click`	`{"action": "click", "coordinate": [x, y]}`
`long_press`	`{"action": "long_press", "coordinate": [x, y]}`
`input_text`	`{"action": "input_text", "text": ""}`
`scroll`	`{"action": "scroll", "direction": "up|down|left|right"}`
`drag`	`{"action": "drag", "coordinate": [x, y], "direction": "up|down|left|right"}`
`navigate_back`	`{"action": "navigate_back"}`
`navigate_home`	`{"action": "navigate_home"}`
`wait`	`{"action": "wait"}`
`status`	`{"action": "status", "goal_status": "complete|infeasible"}`
`answer`	`{"action": "answer", "text": ""}`

Hard Constraints.
- Use `answer` only when the task can be completed without interacting with the device UI.
- Otherwise, use device actions and do not answer directly.
- Output `goal_status="complete"` only after verifying the result on screen.

When to Use Memory. Use `<memory>` only when cross-step storage or comparison is needed; otherwise output `{}`.

Memory Schema. When used, memory must follow this structure:

```
{
"phase": "harvest" | "execute",
"context": {},
"schema": {
"fields": []
},
"items": {
"1": {
"content": {},
"status": "remaining" | "finished" | "skipped",
"skipReason": ""
}
}
}
```

Rules.
- Store only structured values in memory; do not store raw text dumps.

Memory Usage Protocol.
- If using memory, start with `phase="harvest"` and switch to `phase="execute"` once enough information is collected.
- Update an item status to `finished` in the same step after a successful operation.

E.2 Benchmark Task-Generation Prompt

The prompt below guides the generator to vary data values and entry counts while preserving each template's apps, operation types, output structure, and selection constraints. Each generated instance includes a task instruction, initialization data, and expected outputs that are consistent with one another. The count ranges in the prompt guide generation, while Table 8 reports the actual counts for each task family across DC-V0–V3. DC-V0, DC-V1, DC-V2, and DC-V3 use the easy, medium, hard, and very hard configurations, respectively.

System Prompt for Data-Centric Task Variant Generation

Task. Generate feasible GUI automation task variants for an Android long-horizon benchmark.

Inputs.
- `requested_num_tasks`: exact number of variants to output.
- `per_variant_difficulty`: ordered list of difficulty labels.
- `full_benchmark_template`: complete benchmark JSON for one task family.
- Optional trajectory screenshots: UI references only.

Task-Class Constraints. Generate variants for the **same logical task class** as the source template.
- Keep the same apps, cross-app workflow, and output types.
- Scenario wording and concrete values may change, but apps, steps, and structural requirements must not be added or removed.
- `task_goal` and `cn_task_goal` should remain concise, specific, and agent-actionable.

Difficulty Model. Difficulty is increased only through **target data volume** and **confusable distractor density**, not by adding new steps or extra instructions.

Difficulty	Operational / Target Data	Noise / Distractor Data
`easy`	1–2 target sub-groups.	2–5 noise items, clearly distinct from targets.
`medium`	2–3 target sub-groups; roughly 1.5× easy counts.	5–8 noise items with structural similarity.
`hard`	3–5 target sub-groups; roughly 2× easy counts.	8–12 highly similar noise items, including near-duplicate names, formats, or partial-match fields.
`very_hard`	5–8 target sub-groups; roughly 3× easy counts.	12–20 highly confusable noise items, including trap entries with one missing or close-but-wrong field.

Reference and Ground Truth. When an `lh_v1_reference` is provided, use it as the authoritative reference for:
1. `task_goal` and `cn_task_goal` phrasing;
2. `expected_output` structure, including top-level keys, nesting, and field names;
3. `init_data.custom_params` schema.

Only concrete values, counts, and scenario entities may change.

Expected Output Rules.
- Fill `expected_output` to exactly match what a correct agent should produce for the current variant.
- Use the source template's `expected_output_example` as the structural reference.
- Do not emit `rule_validation`; it is derived automatically from `expected_output`.
- Do not emit keys starting with _ inside `expected_output`.

Initialization Data.
- `init_data` must not be null.
- Follow the structural layout of `full_benchmark_template.initialization_example`.
- Ensure unique names, phone numbers, and IDs for all contacts or records.
- Strip all _-prefixed keys from emitted initialization data.

Output Format. Return a single valid JSON object only. Do not include markdown fences, comments, or extra text.

```
{
"task_type": "<exact task type from source template>",
"variants": [
{
"variant_id": 1,
"difficulty": "easy|medium|hard|very_hard",
"task_goal": "<same task class; values may change>",
"cn_task_goal": "<Chinese task goal>",
"init_data": {
"task_type": "<same as task_type>",
"custom_params": { "<seeded data fields from template>" }
},
"expected_output": { "<ground truth with reference structure>" },
"rationale": "<=120 words describing target/noise counts>"
}
]
}
```

```
{
  "task": "CalendarMeetingPrepAndNotify",
  "goal": "Check next week's meetings in Simple Calendar Pro. For each target meeting, find
     related pending tasks in Tasks whose notes mention the meeting title, create a Markor
     agenda file named {meeting_title}_agenda.md, look up attendee phone numbers in Contacts,
      and send the agenda via Simple SMS Messenger.",
  "apps": [
    "Simple Calendar Pro",
    "Tasks",
    "Contacts",
    "Markor",
    "Simple SMS Messenger"
  ],
  "task_type": "long-horizon",

  "_why_hard": [
    "Five apps are chained: Calendar -> Tasks -> Markor -> Contacts -> SMS.",
    "Target meetings, tasks, and contacts are mixed with confusable distractors.",
    "The SMS body must match the corresponding meeting agenda."
  ],

  "task_extension_suggestions": {
    "1": "Increase the number of target meetings; each meeting requires an independent agenda
      file and SMS messages.",
    "2": "Increase noise_events, noise_tasks, and noise_contacts.",
    "3": "Add attendees without matching contacts to test skipping or fault tolerance.",
    "4": "Require strict agenda filename or section formatting."
  },

  "difficulty_control": {
    "easy": {
      "target_entries": 4,
      "same_schema_distractors": 5,
      "required_operations": 3
    },
    "medium": {
      "target_entries": 5,
      "same_schema_distractors": 12,
      "required_operations": 3
    },
    "hard": {
      "target_entries": 20,
      "same_schema_distractors": 43,
      "required_operations": 12
    },
    "very_hard": {
      "target_entries": 36,
      "same_schema_distractors": 100,
      "required_operations": 22
    }
  },
  "custom_params_template": {
    "calendar_events": {
      "target_events": [
        {
          "title": "Vendor Contract Sync",
          "date": "2023-10-24",
          "time": "10:00",
          "duration_mins": 60,
          "location": "Meeting Room 2B",
          "description": "Align on contract renewal timeline and action items.",
          "attendees": ["Nina Patel", "Omar Reyes"]
        }
      ],
      "noise_events": [
        {
          "title": "Team Lunch",
          "date": "2023-10-25",
          "time": "12:00",
          "duration_mins": 60,
          "description": "Lunch at the cafe.",
          "attendees": []
        }
      ]
    },

    "tasks_app_data": {
      "target_tasks": [
        {
          "title": "Draft agenda for vendor sync",
```

```
          "completed": 0,
          "deleted": 0,
          "notes": "For Vendor Contract Sync: propose topics and timing."
        },
        {
          "title": "Collect renewal milestones",
          "completed": 0,
          "deleted": 0,
          "notes": "Needed for Vendor Contract Sync."
        }
      ],
      "noise_tasks": [
        {
          "title": "Buy dish soap",
          "completed": 0,
          "deleted": 0,
          "notes": null
        }
      ]
    },

    "contacts_data": {
      "target_contacts": [
        {
          "name": "Nina Patel",
          "number": "+13661100111"
        },
        {
          "name": "Omar Reyes",
          "number": "+13661100112"
        }
      ],
      "noise_contacts": [
        {
          "name": "Mia Chen",
          "number": "+13661100202"
        }
      ]
    },

    "output_config": {
      "markor_file_template": "{meeting_title}_agenda.md",
      "markor_content_format": "# {meeting_title}\n- Date: {date}\n- Time: {time}\n- Location:
      {location}\n- Attendees: {attendees}\n\n## Related Tasks\n{task_list}",
      "sms_format": "Agenda for {meeting_title} ({date} {time}): {task_list}. Location: {
     location}."
    }
  },

  "initialization_example": {
    "task_type": "CalendarMeetingPrepAndNotify",
    "task_idx": 0,
    "custom_params": {
      "calendar_events": "instantiated from custom_params_template.calendar_events",
      "tasks_app_data": "instantiated from custom_params_template.tasks_app_data",
      "contacts_data": "instantiated from custom_params_template.contacts_data",
      "output_config": "instantiated from custom_params_template.output_config"
    }
  },

  "expected_output_example": {
    "markor_files": [
      {
        "file_name": "Vendor_Contract_Sync_agenda.md",
        "content": "# Vendor Contract Sync\n- Date: 2023-10-24\n- Time: 10:00\n- Location:
     Meeting Room 2B\n- Attendees: Nina Patel, Omar Reyes\n\n## Related Tasks\n- Draft agenda
      for vendor sync\n- Collect renewal milestones\n"
      }
    ],
    "sms_sent": [
      {
        "to": "Nina Patel (+13661100111)",
        "for_meeting": "Vendor Contract Sync",
        "body": "Agenda for Vendor Contract Sync: Draft agenda for vendor sync; Collect
     renewal milestones. Location: Meeting Room 2B."
      },
      {
        "to": "Omar Reyes (+13661100112)",
        "for_meeting": "Vendor Contract Sync",
        "body": "Agenda for Vendor Contract Sync: Draft agenda for vendor sync; Collect
     renewal milestones. Location: Meeting Room 2B."
```

```
        }
      ]
    }
  }
```

Listing 1: Compact example of a data-centric workflow template. Arrays are shortened for readability while preserving the main template structure.